Intel® Integrated Performance Primitives for Intel® Architecture

Quick Reference

Part 4: Cryptography

Document Number: 307151-003

World Wide Web: http://developer.intel.com

Version	Version Information	Date
-001	Original issue. Documents API included into Intel IPP release 5.0 beta. Links to Intel IPP Reference Manual (document number 303881-03).	03/2005
-002	Documents API included into Intel IPP release 5.0 gold. Links to Intel IPP Reference Manual (document number 303881-04).	08/2005
-003	Documents API included into Intel IPP release 5.1 gold. Links to Intel IPP Reference Manual (document number 303881-05).	02/2006

Overview

This Quick Reference provides a full list of prototypes for all functions included into the cryptography domain of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® architecture.

The Intel IPP software is a new generation of the Intel® Performance Libraries, comprising a broad range of functions for basic software functionality and including, among many others, the cryptographic functions domain.

Intel IPP is a cross architecture software library optimized for the latest generations of Intel microprocessors, including Intel® Pentium® 4 processor, Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3), Intel® Xeon® processor, Intel® Xeon® processor with Intel® Extended Memory 64 Technology (Intel® EM64T), Intel® Itanium® 2 processor, and Intel® PCA application processors.

The cryptography subset of Intel IPP includes the following functional domains:

- Symmetric Cryptography Primitive Functions
- One-Way Hash Primitives
- Data Authentication Primitive Functions
- Public Key Cryptography Functions.

This Quick Reference contains an alphabetic list of all cryptography function names with hyperlinks to the full set of function prototypes available in the library and a short description of function operation. The detailed information about all functions, including introductory material, general argument description, function behavior explanation, return values, examples and more, is given in the Intel® Integrated Performance Primitives Reference Manual, volume 4 (document number 303881-05).

Each section and function name included into the reference section of this Quick Reference has a hyperlink to the respective section of the above Reference Manual.

For these links to work properly, please make sure that

- You have the correct version of the Intel IPP Reference Manual available with the Intel IPP distribution (file name *ippcpman.pdf*)
- Files for the Quick Reference and for the Reference Manual reside in the same directory.

If you have jumped to the manual and wish to return to the Quick Reference, use the Go to the Previous View button in the Adobe Acrobat* tool. If you plan to jump between the Quick Reference and the Manual frequently, it is preferable to open the Reference Manual before clicking hyperlinks in the Quick Reference. In this case, you will see both documents in the Window menu of your viewer.

For more information about the Intel IPP library, please refer to the Web site at developer.intel.com/software/products/ipp/.

Alphabetic List of Routines

A

Add_BN
Add_BNU
ARCFourCheckKey
ARCFourDecrypt
ARCFourEncrypt
ARCFourGetSize
ARCFourInit
ARCFourReset

B

BigNumGetSize
BigNumInit
BlowfishDecryptCBC
BlowfishDecryptCFB
BlowfishDecryptCTR
BlowfishDecryptECB
BlowfishEncryptCBC
BlowfishEncryptCFB
BlowfishEncryptCTR
BlowfishEncryptECB
BlowfishGetSize
BlowfishInit

C

Cmp_BN
CmpZero_BN

D

DAABlowfishFinal
DAABlowfishGetSize
DAABlowfishInit
DAABlowfishMessageDigest
DAABlowfishUpdate
DAADESFinal
DAADESGetSize
DAADESInit
DAADESMessageDigest
DAADESUpdate
DAARijndael128Final
DAARijndael128GetSize
DAARijndael128Init
DAARijndael128MessageDigest
DAARijndael128Update
DAARijndael192Final
DAARijndael192GetSize
DAARijndael192Init
DAARijndael192MessageDigest
DAARijndael192Update
DAARijndael256Final
DAARijndael256GetSize
DAARijndael256Init
DAARijndael256MessageDigest
DAARijndael256Update
DAATDESFinal
DAATDESGetSize
DAATDESInit
DAATDESMessageDigest
DAATDESUpdate
DAATwofishFinal
DAATwofishGetSize
DAATwofishInit
DAATwofishMessageDigest
DAATwofishUpdate
DESDecryptCBC
DESDecryptCFB
DESDecryptCTR
DESDecryptECB
DESEncryptCBC
DESEncryptCFB
DESEncryptCTR
DESEncryptECB
DESGetSize
DESInit
Div_64u32u
Div_BN
DLPGenerateDH
DLPGenerateDSA
DLPGenKeyPair
DLPGet

DLPGetDP
DLPGetSize
DLPInit
DLPPublicKey
DLPSet
DLPSetDP
DLPSetKeyPair
DLPSharedSecretDH
DLPSignDSA
DLPValidateDH
DLPValidateDSA
DLPValidateKeyPair
DLPVerifyDSA

E

ECCBAddPoint
ECCBCheckPoint
ECCBComparePoint
ECCBGenKeyPair
ECCBGet
ECCBGetOrderBitSize
ECCBGetPoint
ECCBGetSize
ECCBInit
ECCBMulPointScalar
ECCBNegativePoint
ECCBPointGetSize
ECCBPointInit
ECCBPublicKey
ECCBSet
ECCBSetKeyPair
ECCBSetPoint
ECCBSetPointAtInfinity
ECCBSetStd
ECCBSharedSecretDH
ECCBSharedSecretDHC
ECCBSignDSA
ECCBSignNR
ECCBValidate
ECCBValidateKeyPair
ECCBVerifyDSA
ECCBVerifyNR
ECCPAddPoint
ECCPCheckPoint
ECCPComparePoint
ECCPGenKeyPair
ECCPGet
ECCPGetOrderBitSize
ECCPGetPoint
ECCPGetSize
ECCPInit
ECCPMulPointScalar
ECCPNegativePoint
ECCPPointGetSize
ECCPPointInit
ECCPPublicKey
ECCPSet
ECCPSetKeyPair
ECCPSetPoint
ECCPSetPointAtInfinity
ECCPSetStd
ECCPSharedSecretDH
ECCPSharedSecretDHC
ECCPSignDSA
ECCPSignNR
ECCPValidate
ECCPValidateKeyPair
ECCPVerifyDSA
ECCPVerifyNR

G

Gcd_BN
Get_BN
GetOctString_BN
GetOctString_BNU
GetSize_BN

H

HMACMD5Duplicate
HMACMD5Final
HMACMD5GetSize
HMACMD5Init
HMACMD5MessageDigest
HMACMD5Update

HMACSHA1Duplicate
HMACSHA1Final
HMACSHA1GetSize
HMACSHA1Init
HMACSHA1MessageDigest
HMACSHA1Update
HMACSHA224Duplicate
HMACSHA224Final
HMACSHA224GetSize
HMACSHA224Init
HMACSHA224MessageDigest
HMACSHA224Update
HMACSHA256Duplicate
HMACSHA256Final
HMACSHA256GetSize
HMACSHA256Init
HMACSHA256MessageDigest
HMACSHA256Update
HMACSHA384Duplicate
HMACSHA384Final
HMACSHA384GetSize
HMACSHA384Init
HMACSHA384MessageDigest
HMACSHA384Update
HMACSHA512Duplicate
HMACSHA512Final
HMACSHA512GetSize
HMACSHA512Init
HMACSHA512MessageDigest
HMACSHA512Update

M

MAC_BN_I
MACOne_BNU_I
MD5Duplicate
MD5Final
MD5GetSize
MD5Init
MD5MessageDigest
MD5Update
MGF_MD5
MGF_SHA1
MGF_SHA224
MGF_SHA256
MGF_SHA384
MGF_SHA512
Mod_BN
ModInv_BN
MontExp
MontForm
MontGet
MontGetSize
MontInit
MontMul
MontSet
Mul_BN
Mul_BNU4
Mul_BNU8
MulOne_BNU

P

PrimeGen
PrimeGet
PrimeGet_BN
PrimeGetSize
PrimeInit
PrimeSet
PrimeSet_BN
PrimeTest
PRNGen
PRNGen_BN
PRNGGetSize
PRNGInit
PRNGSetAugment
PRNGSetH0
PRNGSetModulus
PRNGSetSeed

R

Rijndael128DecryptCBC
Rijndael128DecryptCCM
Rijndael128DecryptCFB
Rijndael128DecryptCTR
Rijndael128DecryptECB

Rijndael128EncryptCBC
Rijndael128EncryptCCM
Rijndael128EncryptCFB
Rijndael128EncryptCTR
Rijndael128EncryptECB
Rijndael128GetSize
Rijndael128Init
Rijndael192DecryptCBC
Rijndael192DecryptCFB
Rijndael192DecryptCTR
Rijndael192DecryptECB
Rijndael192EncryptCBC
Rijndael192EncryptCFB
Rijndael192EncryptCTR
Rijndael192EncryptECB
Rijndael192GetSize
Rijndael192Init
Rijndael256DecryptCBC
Rijndael256DecryptCFB
Rijndael256DecryptCTR
Rijndael256DecryptECB
Rijndael256EncryptCBC
Rijndael256EncryptCFB
Rijndael256EncryptCTR
Rijndael256EncryptECB
Rijndael256GetSize
Rijndael256Init
RSADecrypt
RSAEncrypt
RSAGenerate
RSAGetKey
RSAGetSize
RSAInit
RSAOAEPDecrypt
RSAOAEPDecrypt_MD5
RSAOAEPDecrypt_SHA1
RSAOAEPDecrypt_SHA224
RSAOAEPDecrypt_SHA256
RSAOAEPDecrypt_SHA384
RSAOAEPDecrypt_SHA512
RSAOAEPEncrypt
RSAOAEPEncrypt_MD5
RSAOAEPEncrypt_SHA1
RSAOAEPEncrypt_SHA224
RSAOAEPEncrypt_SHA256
RSAOAEPEncrypt_SHA384
RSAOAEPEncrypt_SHA512
RSASetKey
RSASSASign
RSASSASign_MD5
RSASSASign_SHA1
RSASSASign_SHA224
RSASSASign_SHA256
RSASSASign_SHA384
RSASSASign_SHA512
RSASSAVerify
RSASSAVerify_MD5
RSASSAVerify_SHA1
RSASSAVerify_SHA224
RSASSAVerify_SHA256
RSASSAVerify_SHA384
RSASSAVerify_SHA512
RSAValidate

S

Set_BN
SetOctString_BN
SetOctString_BNU
SHA1Duplicate
SHA1Final
SHA1GetSize
SHA1Init
SHA1MessageDigest
SHA1Update
SHA224Duplicate
SHA224Final
SHA224GetSize
SHA224Init
SHA224MessageDigest
SHA224Update
SHA256Duplicate
SHA256Final
SHA256GetSize
SHA256Init

SHA256MessageDigest
SHA256Update
SHA384Duplicate
SHA384Final
SHA384GetSize
SHA384Init
SHA384MessageDigest
SHA384Update
SHA512Duplicate
SHA512Final
SHA512GetSize
SHA512Init
SHA512MessageDigest
SHA512Update
Sqr_32u64u
Sqr_BNU4
Sqr_BNU8
Sub_BN
Sub_BNU

T

TDESDecryptCBC
TDESDecryptCFB
TDESDecryptCTR
TDESDecryptECB
TDESEncryptCBC
TDESEncryptCFB
TDESEncryptCTR
TDESEncryptECB
TwofishDecryptCBC
TwofishDecryptCFB
TwofishDecryptCTR
TwofishDecryptECB
TwofishEncryptCBC
TwofishEncryptCFB
TwofishEncryptCTR
TwofishEncryptECB
TwofishGetSize
TwofishInit

Symmetric Cryptography Primitive Functions

DES/TDES Functions

DESGetSize

Gets the size of the `IppsDESSpec` context.

```
IppStatus ippsDESGetSize(int* pSize)
```

DESInit

Initializes user supplied memory as the `IppsDESSpec` context for future use.

```
IppStatus ippsDESInit(const Ipp8u* pKey, IppsDESSpec* pCtx);
```

DESEncryptECB

Encrypts a variable length data stream in the ECB mode.

```
IppStatus ippsDESEncryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   const IppsDESSpec* pCtx, IppsCPPadding padding);
```

DESDecryptECB

Decrypts a variable length data stream in the ECB mode.

```
IppStatus ippsDESDecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   const IppsDESSpec* pCtx, IppsCPPadding padding);
```

DESEncryptCBC

Encrypts a variable length data stream in the CBC mode.

```
IppStatus ippsDESEncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   const IppsDESSpec* pCtx, Ipp8u* pIV, IppsCPPadding padding);
```

DESDecryptCBC

Decrypts a variable length data stream in the CBC mode.

```
IppStatus ippsDESDecryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   const IppsDESSpec* pCtx, Ipp8u* pIV, IppsCPPadding padding);
```

DESEncryptCFB

Encrypts a variable length data stream in the CFB mode.

```
IppStatus ippsDESEncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int srcLen,
   int cfbBlkSize, const IppsDESSpec* pCtx, Ipp8u* pIV, IppsCPPadding
   padding);
```

DESDecryptCFB

Decrypts a variable length data stream in the CFB mode.

```
IppStatus ippsDESDecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   int cfbBlkSize, const IppsDESSpec* pCtx, const Ipp8u* pIV,
   IppsCPPadding padding);
```

DESEncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsDESEncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int srcLen,
   const IppsDESSpec* pCtx, Ipp8u* pCtrValue, int ctrNumBitSize);
```

DESDecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsDESDecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int srcLen,
   const IppsDESSpec* pCtx, Ipp8u* pCtrValue, int ctrNumBitSize);
```

TDESEncryptECB

Encrypts variable length data stream in ECB mode.

```
IppStatus ippsTDESEncryptECB(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
   *pCtx3, IppsCPPadding padding);
```

TDESDecryptECB

Decrypts variable length data stream in the ECB mode.

```
IppStatus ippsTDESDecryptECB(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
   *pCtx3, IppsCPPadding padding);
```

TDESEncryptCBC

Encrypts variable length data stream in the CBC mode.

```
IppStatus ippsTDESEncryptCBC(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
   *pCtx3, Ipp8u *pIV, IppsCPPadding padding);
```

TDESDecryptCBC

Decrypts variable length data stream in the CBC mode.

```
IppStatus ippsTDESDecryptCBC(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
   *pCtx3, Ipp8u *pIV, IppsCPPadding padding);
```

TDESEncryptCFB

Encrypts variable length data stream in the CFB mode.

```
IppStatus ippsTDESEncryptCFB(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   int cfbBlkSize, const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2,
   const IppsDESSpec *pCtx3, Ipp8u *pIV, IppsCPPadding padding);
```

TDESDecryptCFB

Decrypts variable length data stream in the CFB mode.

```
IppStatus ippsTDESDecryptCFB(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
   int cfbBlkSize, const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2,
   const IppsDESSpec *pCtx3, Ipp8u *pIV, IppsCPPadding padding);
```

TDESEncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsTDESEncryptCTR(const Ipp8u *pSrc, Ipp8u *pDst, int srclen,
    const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
    *pCtx3, Ipp8u *pCtrValue, int ctrNumBitSize);
```

TDESDecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsTDESDecryptCTR(const Ipp8u *pSrc, Ipp8u *pDst, int srcLen,
    const IppsDESSpec *pCtx1, const IppsDESSpec *pCtx2, const IppsDESSpec
    *pCtx3, Ipp8u *pCtrValue, int ctrNumBitSize);
```

Rijndael Functions

Rijndael128GetSize

Gets the size of the `IppsRijndael128Spec` context.

```
IppStatus ippsRijndael128GetSize(int* pSize)
```

Rijndael128Init

Initializes user supplied memory as `IppsRijndael128Spec` context for future use.

```
IppStatus ippsRijndael128Init(const Ipp8u *pKey, IppsRijndaelKeyLength
    keylen, IppsRijndael128Spec* pCtx);
```

Rijndael128EncryptECB

Encrypts plaintext message by using ECB encryption mode.

```
IppStatus ippsRijndael128EncryptECB(const Ipp8u  *pSrc, Ipp8u  *pDst, int
    srclen, const IppsRijndael128Spec* pCtx, IppsCPPadding padding);
```

Rijndael128DecryptECB

Decrypts byte data stream by using Rijndael algorithm in the ECB mode.

```
IppStatus ippsRijndael128DecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsRijndael128Spec* pCtx, IppsCPPadding padding);
```

Rijndael128EncryptCBC

Encrypts byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael128EncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsRijndael128Spec* pCtx, const Ipp8u* pIV,
    IppsCPPadding padding);
```

Rijndael128DecryptCBC

Decrypts byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael128DecryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsRijndael128Spec* pCtx, const Ipp8u* pIV,
    IppsCPPadding padding);
```

Rijndael128EncryptCFB

Encrypts byte data stream according to Rijndael in the CFB mode.

```
IppStatus ippsRijndael128EncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, int cfbBlkSize, const IppsRijndael128Spec* pCtx, const Ipp8u
   *pIV, IppsCPPadding padding);
```

Rijndael128DecryptCFB

Decrypts byte data stream according to Rijndael in CFB mode.

```
IppStatus ippsRijndael128DecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen,  int cfbBlkSize, const IppsRijndael128Spec* pCtx, const
   Ipp8u* pIV, IppsCPPadding padding);
```

Rijndael128EncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael128EncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsRijndael128Spec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

Rijndael128DecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael128DecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsRijndael128Spec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

Rijndael128EncryptCCM

Encrypts a variable length data stream and generates its authentication tag in the CCM mode.

```
IppStatus ippsRijndael128EncryptCCM(const Ipp8u* pNonce, int nonceLen,
   const Ipp8u* pAssc, int asscLen, const Ipp8u* pSrc, int srcLen, int
   macLen, Ipp8u* pDst, const IppsRijndael128Spec* pCtx);
```

Rijndael128DecryptCCM

Decrypts and verifies a variable length data stream in the CCM mode.

```
IppStatus ippsRijndael128DecryptCCM(const Ipp8u* pNonce, int nonceLen,
   const Ipp8u* pAssc, int asscLen, const Ipp8u* pSrc, int srcLen, int
   macLen, Ipp8u* pDst, IppBool* pResult, const IppsRijndael128Spec*
   pCtx);
```

Rijndael192GetSize

Gets the size of the `IppsRijndael192Spec` context.

```
IppStatus ippsRijndael192GetSize(int* pSize)
```

Rijndael192Init

Initializes user supplied memory as the `IppsRijndael192Spec` context for future use.

```
IppStatus ippsRijndael192Init(const Ipp8u *pKey, IppsRijndaelKeyLength
   keylen, IppsRijndael192Spec* pCtx);
```

Rijndael192EncryptECB

Encrypts a byte data stream according to Rijndael in the ECB mode.

```
IppStatus ippsRijndael192EncryptECB(const Ipp8u  *pSrc, Ipp8u  *pDst, int
   srclen, const IppsRijndael192Spec* pCtx, IppsCPPadding padding);
```

Rijndael192DecryptECB

Decrypts byte data stream according to Rijndael in the EBC mode.

```
IppStatus ippsRijndael192DecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael192Spec *pCtx, IppsCPPadding padding);
```

Rijndael192EncryptCBC

Encrypts a byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael192EncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael192Spec* pCtx, const Ipp8u* pIV,
   IppsCPPadding padding);
```

Rijndael192DecryptCBC

Decrypts a byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael192DecryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael192Spec* pCtx, const Ipp8u* pIV,
   IppsCPPadding padding);
```

Rijndael192EncryptCFB

Encrypts a byte data stream according to Rijndael in the CFB mode.

```
IppStatus ippsRijndael192EncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, int cfbBlkSize, const IppsRijndael192Spec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

Rijndael192DecryptCFB

Decrypts a byte data stream according to Rijndael in the CFB mode.

```
IppStatus ippsRijndael192DecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, int cfbBlkSize, const IppsRijndael192Spec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

Rijndael192EncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael192EncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsRijndael192Spec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

Rijndael192DecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael192DecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsRijndael192Spec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

Rijndael256GetSize

Gets the size of the `IppsRijndael256Spec` context.

```
IppStatus ippsRijndael256GetSize(int* pSize)
```

Rijndael256Init

Initializes user supplied memory as `IppsRijndael256Spec` context for future use.

```
IppStatus ippsRijndael256Init(const Ipp8u *pKey, IppsRijndaelKeyLength
   keylen, IppsRijndael256Spec* pCtx);
```

Rijndael256EncryptECB

Encrypts a byte data stream according to Rijndael in the ECB mode.

```
IppStatus ippsRijndael256EncryptECB(const Ipp8u  *pSrc, Ipp8u  *pDst, int
   srclen, const IppsRijndael256Spec* pCtx, IppsCPPadding padding);
```

Rijndael256DecryptECB

Decrypts a byte data stream according to Rijndael in the ECB mode.

```
IppStatus ippsRijndael256DecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael256Spec* pCtx, IppsCPPadding padding);
```

Rijndael256EncryptCBC

Encrypts byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael256EncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael256Spec* pCtx, const Ipp8u* pIV,
   IppsCPPadding padding);
```

Rijndael256DecryptCBC

Decrypts byte data stream according to Rijndael in the CBC mode.

```
IppStatus ippsRijndael256DecryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsRijndael256Spec* pCtx, const Ipp8u* pIV,
   IppsCPPadding padding);
```

Rijndael256EncryptCFB

Encrypts byte data stream according to Rijndael in the CFB mode.

```
IppStatus ippsRijndael256EncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, int cfbBlkSize, const IppsRijndael256Spec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

Rijndael256DecryptCFB

Decrypts byte data stream according to Rijndael in the CFB mode.

```
IppStatus ippsRijndael256DecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, int cfbBlkSize, const IppsRijndael256Spec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

Rijndael256EncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael256EncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
    srcLen, const IppsRijndael256Spec* pCtx, Ipp8u* pCtrValue, int
    ctrNumBitSize);
```

Rijndael256DecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsRijndael256DecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
    srcLen, const IppsRijndael256Spec* pCtx, Ipp8u* pCtrValue, int
    ctrNumBitSize);
```

Blowfish Functions

BlowfishGetSize

Gets the size of the `IppsBlowfishSpec` context.

```
IppStatus ippsBlowfishGetSize(int* pSize)
```

BlowfishInit

Initializes user supplied memory as the `IppsBlowfishSpec` context for future use.

```
IppStatus ippsBlowfishInit(const Ipp8u *pKey, int keylen,
    IppsBlowfishSpec* pCtx);
```

BlowfishEncryptECB

Encrypts input plaintext in the ECB mode.

```
IppStatus ippsBlowfishEncryptECB(const Ipp8u *pSrc, Ipp8u *pDst, int
    scrlen, const IppsBlowfishSpec* pCtx, IppsCPPadding padding);
```

BlowfishDecryptECB

Decrypts byte data stream according to Blowfish scheme in the EBC mode.

```
IppStatus ippsBlowfishDecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsBlowfishSpec* pCtx, IppsCPPadding padding);
```

BlowfishEncryptCBC

Encrypts byte data stream according to Blowfish scheme in the CBC mode.

```
IppStatus ippsBlowfishEncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsBlowfishSpec* pCtx, const Ipp8u* pIV,
    IppsCPPadding padding);
```

BlowfishDecryptCBC

Decrypts byte data stream according to Blowfish scheme in the CBC mode.

```
IppStatus ippsBlowfishDecryptCBC(const  Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsBlowfishSpec* pCtx, const Ipp8u* pIV,
    IppsCPPadding padding);
```

BlowfishEncryptCFB

Encrypts byte data stream according to Blowfish scheme in the CFB mode.

```
IppStatus ippsBlowfishEncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, int cfbBlkSize, const IppsBlowfishSpec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

BlowfishDecryptCFB

Decrypts byte data stream according to Blowfish scheme in the CFB mode.

```
IppStatus ippsBlowfishDecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, int cfbBlkSize, const IppsBlowfishSpec* pCtx, const Ipp8u*
   pIV, IppsCPPadding padding);
```

BlowfishEncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsBlowfishEncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsBlowfishSpec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

BlowfishDecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsBlowfishDecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
   srcLen, const IppsBlowfishSpec* pCtx, Ipp8u* pCtrValue, int
   ctrNumBitSize);
```

Twofish Functions

TwofishGetSize

Gets the size of the `IppsTwofishSpec` context.

```
IppStatus ippsTwofishGetSize(int* pSize)
```

TwofishInit

Initializes user supplied memory as the `IppsTwofishSpec` context for future use.

```
IppStatus ippsTwofishInit(const Ipp8u *pKey, int keylen,
   IppsTwofishSpec* pCtx);
```

TwofishEncryptECB

Encrypts input plaintext in the ECB mode.

```
IppStatus ippsTwofishEncryptECB(const Ipp8u *pSrc, Ipp8u *pDst, int
   scrlen, const IppsTwofishSpec* pCtx, IppsCPPadding padding);
```

TwofishDecryptECB

Decrypts byte data stream according to Twofish scheme in the EBC mode.

```
IppStatus ippsTwofishDecryptECB(const Ipp8u* pSrc, Ipp8u* pDst, int
   srclen, const IppsTwofishSpec* pCtx, IppsCPPadding padding);
```

TwofishEncryptCBC

Encrypts byte data stream according to Twofish scheme in the CBC mode.

```
IppStatus ippsTwofishEncryptCBC(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, const IppsTwofishSpec* pCtx, const Ipp8u* pIV,  IppsCPPadding
    padding);
```

TwofishDecryptCBC

Decrypts byte data stream according to Twofish scheme in the CBC mode.

```
IppStatus ippsTwofishDecryptCBC(const  Ipp8u* pSrc, Ipp8u *pDst, int
    srclen, const IppsTwofishSpec* pCtx, const Ipp8u* pIV,  IppsCPPadding
    padding);
```

TwofishEncryptCFB

Encrypts byte data stream according to Twofish scheme in the CFB mode.

```
IppStatus ippsTwofishEncryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
    srcLen, int cfbBlkSize, const IppsTwofishSpec* pCtx,  const Ipp8u*
    pIV, IppsCPPadding padding);
```

TwofishDecryptCFB

Decrypts byte data stream according to Twofish scheme in the CFB mode.

```
IppStatus ippsTwofishDecryptCFB(const Ipp8u* pSrc, Ipp8u* pDst, int
    srclen, int cfbBlkSize, const IppsTwofishSpec* pCtx, const Ipp8u*
    pIV, IppsCPPadding padding);
```

TwofishEncryptCTR

Encrypts a variable length data stream in the CTR mode.

```
IppStatus ippsTwofishEncryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
    srcLen, const IppsTwofishSpec* pCtx, Ipp8u* pCtrValue, int
    ctrNumBitSize);
```

TwofishDecryptCTR

Decrypts a variable length data stream in the CTR mode.

```
IppStatus ippsTwofishDecryptCTR(const Ipp8u* pSrc, Ipp8u* pDst, int
    srcLen, const IppsTwofishSpec* pCtx, Ipp8u* pCtrValue, int
    ctrNumBitSize);
```

ARCFour Functions

ARCFourGetSize

Gets the size of the `IppsARCFourState` context.

```
IppStatus ippsARCFourGetSize(int* pSize);
```

ARCFourCheckKey

Checks weakness of a user-defined key.

```
IppStatus ippsARCFourCheckKey(const Ipp8u* pKey, int keyLen, IppsBool*
   pIsWeak);
```

ARCFourInit

Initializes user-supplied memory as the `IppsARCFourState` context for future use.

```
IppStatus ippsARCFourInit(const Ipp8u* pKey, int keyLen,
   IppsARCFourState* pCtx);
```

ARCFourEncrypt

Encrypts a variable length data stream according to ARCFour.

```
IppStatus ippsARCFourEncrypt(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   IppsARCFourState* pCtx);
```

ARCFourDecrypt

Decrypts a variable length data stream according to ARCFour.

```
IppStatus ippsARCFourDecrypt(const Ipp8u* pSrc, Ipp8u* pDst, int srclen,
   IppsARCFourState* pCtx);
```

ARCFourReset

Resets the `IppsARCFourState` context to the initial state.

```
IppStatus ippsARCFourReset(IppsARCFourState* pCtx);
```

One-Way Hash Primitives

Hash Functions

MD5GetSize

Gets the size of the `IppsMD5State` context in bytes.

```
IppStatus ippsMD5GetSize(int *pSize);
```

MD5Init

Initializes user supplied memory as `IppsMD5State` context for future use.

```
IppStatus ippsMD5Init(IppsMD5State* pCtx);
```

MD5Duplicate

Copies one `IppsMD5State` context to another.

```
IppStatus ippsMD5Duplicate(const IppsMD5State* pSrcCtx, IppsMD5State*
   pDstCtx)
```

MD5Update

Digests the current input message stream of the specified length.

```
IppStatus ippsMD5Update(const Ipp8u *pSrcMesg, int mesglen, IppsMD5State
   *pCtx);
```

MD5Final

Completes computation of the MD5 digest value.

```
IppStatus ippsMD5Final(Ipp8u *pMD, IppsMD5State *pCtx);
```

SHA1GetSize

Gets the size of the `IppsSHA1State` context in bytes.

```
IppStatus ippsSHA1GetSize(int *pSize);
```

SHA1Init

Initializes user supplied memory as `IppsSHA1State` context for future use.

```
IppStatus ippsSHA1Init(IppsSHA1State* pCtx);
```

SHA1Duplicate

Copies one `IppsSHA1State` context to another.

```
IppStatus ippsSHA1Duplicate(const IppsSHA1State* pSrcCtx, IppsSHA1State*
   pDstCtx)
```

SHA1Update

Digests the current input message stream of the specified length.

```
IppStatus ippsSHA1Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsSHA1State *pCtx);
```

SHA1Final

Completes computation of the SHA1 digest value.

```
IppStatus ippsSHA1Final(Ipp8u *pMD, IppsSHA1State *pCtx);
```

SHA224GetSize

Gets the size of the `IppsSHA224State` context in bytes.

```
IppStatus ippsSHA224GetSize(int *pSize);
```

SHA224Init

Initializes user supplied memory as `IppsSHA224State` context for future use.

```
IppStatus ippsSHA224Init(IppsSHA224State* pCtx);
```

SHA224Duplicate

Copies one `IppsSHA224State` context to another.

```
IppStatus ippsSHA224Duplicate(const IppsSHA224State* pSrcCtx,
   IppsSHA224State* pDstCtx)
```

SHA224Update

Digests the current input message stream of the specified length.

```
IppStatus ippsSHA224Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsSHA224State *pCtx);
```

SHA224Final

Completes computation of the SHA224 digest value.

```
IppStatus ippsSHA224Final(Ipp8u *pMD, IppsSHA224State *pCtx);
```

SHA256GetSize

Gets the size of the `IppsSHA256State` context in bytes.

```
IppStatus ippsSHA256GetSize(int *pSize);
```

SHA256Init

Initializes user supplied memory as `IppsSHA256State` context for future use.

```
IppStatus ippsSHA256Init(IppsSHA256State *pCtx);
```

SHA256Duplicate

Copies one `IppsSHA256State` context to another.

```
IppStatus ippsSHA256Duplicate(const IppsSHA256State* pSrcCtx,
   IppsSHA256* pDstCtx)
```

SHA256Update

Digests the current input message stream of the specified length.

```
IppStatus ippsSHA256Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsSHA256State *pCtx);
```

SHA256Final

Completes computation of the SHA256 digest value.

```
IppStatus ippsSHA256Final(Ipp8u *pMD, IppsSHA256State *pCtx);
```

SHA384GetSize

Gets the size of the `IppsSHA384State` context in bytes.

```
IppStatus ippsSHA384GetSize(int *pSize);
```

SHA384Init

Initializes user supplied memory as `IppsSHA384State` context for future use.

```
IppStatus ippsSHA384Init(IppsSHA384State* pCtx);
```

SHA384Duplicate

Copies one `IppsSHA384State` context to another.

```
IppStatus ippsSHA384Duplicate(const IppsSHA384State* pSrcCtx,
   IppsSHA384State* pDstCtx)
```

SHA384Update
Digests the current input message stream of the specified length.

```
IppStatus ippsSHA384Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsSHA384State *pCtx);
```

SHA384Final
Completes computing of the SHA384 digest value.

```
IppStatus ippsSHA384Final(Ipp8u *pMD, IppsSHA384State *pCtx );
```

SHA512GetSize
Gets the size of the `IppsSHA512State` context in bytes.

```
IppStatus ippsSHA512GetSize(int *pSize);
```

SHA512Init
Initializes user supplied memory as `IppsSHA512State` context for future use.

```
IppStatus ippsSHA512Init(IppsSHA512State* pState);
```

SHA512Duplicate
Copies one `IppsSHA512State` context to another.

```
IppStatus ippsSHA512Duplicate(const IppsSHA512State* pSrcCtx,
   IppsSHA512* pDstCtx)
```

SHA512Update
Digests the current input message stream of the specified length.

```
IppStatus ippsSHA512Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsSHA512State *pCtx);
```

SHA512Final
Completes computation of the SHA512 digest value.

```
IppStatus ippsSHA512Final(Ipp8u *pMD, IppsSHA512State *pCtx);
```

Generalized Hash Functions for Non-Streaming Messages

MD5MessageDigest
Computes MD5 digest value of the input message.

```
IppStatus ippsMD5MessageDigest(const Ipp8u *pSrcMesg, int mesgLen, Ipp8u
   *pMD);
```

SHA1MessageDigest
Computes SHA-1 digest value of the input message.

```
IppStatus ippsSHA1MessageDigest(const Ipp8u *pSrcMesg, int mesglen,
   Ipp8u *pMD);
```

SHA224MessageDigest

Computes SHA-224 digest value of the input message.

```
IppStatus ippsSHA224MessageDigest(const Ipp8u *pSrcMsg, int mesglen,
   Ipp8u *pMD);
```

SHA256MessageDigest

Computes SHA-256 digest value of the input message.

```
IppStatus ippsSHA256MessageDigest(const Ipp8u *pSrcMsg, int mesglen,
   Ipp8u *pMD);
```

SHA384MessageDigest

Computes SHA-384 digest value of the input message.

```
IppStatus ippsSHA384MessageDigest(const Ipp8u *pSrcMsg, int mesglen,
   Ipp8u *pMD);
```

SHA512MessageDigest

Computes SHA-512 digest value of the input message.

```
IppStatus ippsSHA512MessageDigest(const Ipp8u *pSrcMsg, int mesglen,
   Ipp8u *pMD);
```

Mask Generation Functions

MGF_MD5

Generates a pseudorandom mask of the specified length using MD5 hash function.

```
IppStatus ippsMGF_MD5(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask, int
   maskLen);
```

MGF_SHA1

Generates a pseudorandom mask of the specified length using SHA-1 hash function.

```
IppStatus ippsMGF_SHA1(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask, int
   maskLen);
```

MGF_SHA224

Generates a pseudorandom mask of the specified length using SHA-224 hash function.

```
IppStatus ippsMGF_SHA224(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask,
   int maskLen);
```

MGF_SHA256

Generates a pseudorandom mask of the specified length using SHA-256 hash function.

```
IppStatus ippsMGF_SHA256(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask,
   int maskLen);
```

MGF_SHA384

Generates a pseudorandom mask of the specified length using SHA-384 hash function.

```
IppStatus ippsMGF_SHA384(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask,
    int maskLen);
```

MGF_SHA512

Generates a pseudorandom mask of the specified length using SHA-512 hash function.

```
IppStatus ippsMGF_SHA512(const Ipp8u *pSeed, int seedLen, Ipp8u* pMask,
    int maskLen);
```

Data Authentication Primitive Functions

Keyed Hash Functions

HMACSHA1GetSize

Gets the size of the `IppsHMACSHA1State` context.

```
IppStatus ippsHMACSHA1GetSize(int *pSize);
```

HMACSHA1Init

Initializes user supplied memory as `IppsHMACSHA1State` context for future use.

```
IppStatus ippsHMACSHA1Init(const Ipp8u *pKey, int keyLen,
    IppsHMACSHA1State *pCtx;
```

HMACSHA1Duplicate

Copies one `IppsHMACSHA1State` context to another.

```
IppStatus ippsHMACSHA1Duplicate(const IppsHMACSHA1State* pSrcCtx,
    IppsHMACSHA1State* pDstCtx);
```

HMACSHA1Update

Digests the current input message stream of the specified length.

```
IppStatus ippsHMACSHA1Update(const Ipp8u *pSrcMesg, int mesglen,
    IppsHMACSHA1State *pCtx);
```

HMACSHA1Final

Completes computation of the HMAC value.

```
IppStatus ippsHMACSHA1Final(Ipp8u *pMAC, int macLen, IppsHMACSHA1State
    *pCtx);
```

HMACSHA1MessageDigest

Computes the HMAC value of the message.

```
IppStatus ippsHMACSHA1MessageDigest(const Ipp8u *pSrcMesg, int mesgLen,
    const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

HMACSHA224GetSize

Gets the size of the `IppsHMACSHA224State` context.

```
IppStatus ippsHMACSHA224GetSize(int *pSize);
```

HMACSHA224Init

Initializes user supplied memory as `IppsHMACSHA224State` context for future use.

```
IppStatus ippsHMACSHA224Init(const Ipp8u *pKey, int keyLen,
   IppsHMACSHA224State *pCtx);
```

HMACSHA224Duplicate

Copies one `IppsHMACSHA224State` context to another.

```
IppStatus ippsHMACSHA224Duplicate(const IppsHMACSHA224State* pSrcCtx,
   IppsHMACSHA224State* pDstCtx);
```

HMACSHA224Update

Digests the current input message stream of the specified length.

```
IppStatus ippsHMACSHA224Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsHMACSHA224State *pCtx);
```

HMACSHA224Final

Completes computation of the HMAC value.

```
IppStatus ippsHMACSHA224Final(Ipp8u *pMAC, int macLen,
   IppsHMACSHA224State *pCtx);
```

HMACSHA224MessageDigest

Computes the HMAC value of the message.

```
IppStatus ippsHMACSHA224MessageDigest(const Ipp8u *pSrcMesg, int
   mesgLen, const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

HMACSHA256GetSize

Gets the size of the `IppsHMACSHA256State` context.

```
IppStatus ippsHMACSHA256GetSize(int *pSize);
```

HMACSHA256Init

Initializes user supplied memory as `IppsHMACSHA256State` context for future use.

```
IppStatus ippsHMACSHA256Init(const Ipp8u *pKey, int keyLen,
   IppsHMACSHA256State *pCtx);
```

HMACSHA256Duplicate

Copies one `IppsHMACSHA256State` context to another.

```
IppStatus ippsHMACSHA256Duplicate(const IppsHMACSHA256State* pSrcCtx,
   IppsHMACSHA256State* pDstCtx);
```

HMACSHA256Update

Digests the current input message stream of the specified length.

```
IppStatus ippsHMACSHA256Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsHMACSHA256State *pCtx);
```

HMACSHA256Final

Completes computation of the HMAC value.

```
IppStatus ippsHMACSHA256Final(Ipp8u *pMAC, int macLen,
   IppsHMACSHA256State *pCtx);
```

HMACSHA256MessageDigest

Computes the HMAC value of the message.

```
IppStatus ippsHMACSHA256MessageDigest(const Ipp8u *pSrcMesg, int
   mesgLen, const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

HMACSHA384GetSize

Gets the size of the `IppsHMACSHA384State` context.

```
IppStatus ippsHMACSHA384GetSize(int *pSize);
```

HMACSHA384Init

Initializes user supplied memory as `IppsHMACSHA384State` context for future use.

```
IppStatus ippsHMACSHA384Init(const Ipp8u *pKey, int keyLen,
   IppsHMACSHA384State *pCtx);
```

HMACSHA384Duplicate

Copies one `IppsHMACSHA384State` context to another.

```
IppStatus ippsHMACSHA384Duplicate(const IppsHMACSHA384State* pSrcCtx,
   IppsHMACSHA384State* pDstCtx);
```

HMACSHA384Update

Digests the current input message stream of the specified length.

```
IppStatus ippsHMACSHA384Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsHMACSHA384State *pCtx);
```

HMACSHA384Final

Completes computation of the HMAC value.

```
IppStatus ippsHMACSHA384Final(Ipp8u *pMAC, int macLen,
   IppsHMACSHA384State *pCtx);
```

HMACSHA384MessageDigest

Computes the HMAC value of the message.

```
IppStatus ippsHMACSHA384MessageDigest(const Ipp8u *pSrcMesg, int
   mesgLen, const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

HMACSHA512GetSize

Gets the size of the `IppsHMACSHA512State` context.

```
IppStatus ippsHMACSHA512GetSize(int *pSize);
```

HMACSHA512Init

Initializes user supplied memory as `IppsHMACSHA512State` context for future use.

```
IppStatus ippsHMACSHA512Init(const Ipp8u *pKey, int keyLen,
   IppsHMACSHA512State *pCtx);
```

HMACSHA512Duplicate

Copies one `IppsHMACSHA512State` context to another.

```
IppStatus ippsHMACSHA512Duplicate(const IppsHMACSHA512State* pSrcCtx,
   IppsHMACSHA512State* pDstCtx);
```

HMACSHA512Update

Digests the current input message stream of the specified length.

```
IppStatus ippsHMACSHA512Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsHMACSHA512State *pCtx);
```

HMACSHA512Final

Completes computation of the HMAC value.

```
IppStatus ippsHMACSHA512Final(Ipp8u *pMAC, int macLen,
   IppsHMACSHA512State *pCtx);
```

HMACSHA512MessageDigest

Computes the HMAC value of the message.

```
IppStatus ippsHMACSHA512MessageDigest(const Ipp8u *pSrcMesg, int
   mesgLen, const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

HMACMD5GetSize

Gets the size of the `IppsHMACMD5State` context.

```
IppStatus ippsHMACMD5GetSize(int *pSize);
```

HMACMD5Init

Initializes user supplied memory as `IppsHMACMD5State` context for future use.

```
IppStatus ippsHMACMD5Init(const Ipp8u *pKey, int keyLen,
   IppsHMACMD5State *pCtx);
```

HMACMD5Duplicate

Copies one `IppsHMACMD5State` context to another.

```
IppStatus ippsHMACMD5Duplicate(const IppsHMACMD5State* pSrcCtx,
   IppsHMACMD5State* pDstCtx);
```

HMACMD5Update
Digests the current input message stream of the specified length.

```
IppStatus ippsHMACMD5Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsHMACMD5State *pCtx);
```

HMACMD5Final
Completes computation of the HMAC value.

```
IppStatus ippsHMACMD5Final(Ipp8u *pMAC, int macLen, IppsHMACMD5State
   *pCtx);
```

HMACMD5MessageDigest
Computes the HMAC value of the message.

```
IppStatus ippsHMACMD5MessageDigest(const Ipp8u *pSrcMesg, int mesgLen,
   const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

Data Authentication Functions

DAADESGetSize
Gets the size of the `IppsDAADESState` context.

```
IppStatus ippsDAADESGetSize(int *pSize);
```

DAADESInit
Initializes user supplied memory as `IppsDAADESState` context for future use.

```
IppStatus ippsDAADESInit(const Ipp8u *pKey, IppsDAADESState *pCtx);
```

DAADESUpdate
Digests the current input message stream of the specified length.

```
IppStatus ippsDAADESUpdate(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAADESState* pCtx);
```

DAADESFinal
Completes computation of the DAC value.

```
IppStatus ippsDAADESFinal(Ipp8u *pDAC, int dacLen, IppsDAADESState*
   pCtx);
```

DAADESMessageDigest
Computes the DAC value of the message.

```
IppStatus ippsDAADESMessageDigest(const Ipp8u *pSrcMsg, int msgLen,
   const Ipp8u *pKey, Ipp8u *pMAC, int macLen);
```

DAATDESGetSize
Gets the size of the `IppsDAATDESState` context.

```
IppStatus ippsDAATDESGetSize(int *pSize);
```

DAATDESInit

Initializes user supplied memory as the `IppsDAATDESState` context for future use.

```
IppStatus ippsDAATDESInit(const Ipp8u* pKey, const Ipp8u* pKey2, const
   Ipp8u* pKey3, IppsDAATDESState* pCtx);
```

DAATDESUpdate

Digests the current input message stream of the specified length.

```
IppStatus ippsDAATDESUpdate(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAATDESState *pCtx);
```

DAATDESFinal

Completes computation of the DAC value.

```
IppStatus ippsDAATDESFinal(Ipp8u *pDAC, int dacLen, IppsDAATDESState*
   pCtx);
```

DAATDESMessageDigest

Computes the DAC value of the message.

```
IppStatus ippsDAATDESMessageDigest(const Ipp8u *pSrcMsg, int msgLen,
   const Ipp8u *pKey1, const Ipp8u *pKey2, const Ipp8u *pKey3, Ipp8u
   *pMAC, int macLen);
```

DAARijndael128GetSize

Gets the size of the `IppsDAARijndael128State` context.

```
IppStatus ippsDAARijndael128GetSize(int *pSize);
```

DAARijndael128Init

Initializes user supplied memory as `IppsDAARijndael128State` context for future use.

```
IppStatus ippsDAARijndael128Init(const Ipp8u* pKey,
   IppsRijndaelKeyLength keyLen, IppsDAARijndael128State* pCtx);
```

DAARijndael128Update

Digest the current input message stream of the specified length.

```
IppStatus ippsDAARijndael128Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAARijndael128State* pCtx);
```

DAARijndael128Final

Completes computation of the DAC value.

```
IppStatus ippsDAARijndael128Final(Ipp8u *pDAC, int dacLen,
   IppsDAARijndael128State *pCtx);
```

DAARijndael128MessageDigest

Computes the DAC value of the message.

```
IppStatus ippsDAARijndael128MessageDigest(const Ipp8u *pSrcMsg, int
   msgLen, const Ipp8u *pKey, IppsRijndaelKeyLength keyLen, Ipp8u *pMAC,
   int macLen);
```

DAARijndael192GetSize

Gets the size of the `IppsDAARijndael192State` context.

```
IppStatus ippsDAARijndael192GetSize(int *pSize);
```

DAARijndael192Init

Initializes user supplied memory as `IppsDAARijndael192State` context for future use.

```
IppStatus ippsRijndael192Init(const Ipp8u* pKey,
   IppsDAARijndaelKeyLength keyLen, IppsRijndael192State* pCtx);
```

DAARijndael192Update

Digest the current input message stream of the specified length.

```
IppStatus ippsDAARijndael192Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAARijndael192State* pCtx);
```

DAARijndael192Final

Completes computation of the DAC value.

```
IppStatus ippsDAARijndael192Final(Ipp8u *pDAC, int dacLen,
   IppsDAARijndael192State *pCtx);
```

DAARijndael192MessageDigest

Computes the DAC value of the message.

```
IppStatus ippsDAARijndael192MessageDigest(const Ipp8u *pSrcMsg, int
   msgLen, const Ipp8u *pKey, IppsRijndaelKeyLength keyLen, Ipp8u *pDAC,
   int macLen);
```

DAARijndael256GetSize

Gets the size of the `IppsDAARijndael256State` context.

```
IppStatus ippsDAARijndael256GetSize(int *pSize);
```

DAARijndael256Init

Initializes user supplied memory as `IppsDAARijndael256State` context for future use.

```
IppStatus ippsRijndael256Init(const Ipp8u* pKey,
   IppsDAARijndaelKeyLength keyLen, IppsRijndael256State* pCtx);
```

DAARijndael256Update

Digest the current input message stream of the specified length.

```
IppStatus ippsDAARijndael256Update(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAARijndael256State* pCtx);
```

DAARijndael256Final

Completes computation of the DAC value.

```
IppStatus ippsDAARijndael256Final(Ipp8u *pDAC, int dacLen,
   IppsDAARijndael256State* pCtx);
```

DAARijndael256MessageDigest

Computes the DAC value of the message.

```
IppStatus ippsDAARijndael256MessageDigest(const Ipp8u *pSrcMsg, int
   msgLen, const Ipp8u *pKey, IppsRijndaelKeyLength keyLen, Ipp8u *pMAC,
   int maclen);
```

DAABlowfishGetSize

Gets the size of the `IppsDAABlowfishState` context.

```
IppStatus ippsDAABlowfishGetSize(int *pSize);
```

DAABlowfishInit

Initializes user supplied memory as `IppsDAABlowfishState` context for future use.

```
IppStatus ippsDAABlowfishInit(const Ipp8u* pKey, int keylen,
   IppsDAABlowfishState* pCtx);
```

DAABlowfishUpdate

Digests the current input message stream of the specified length.

```
IppStatus ippsDAABlowfishUpdate(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAABlowfishState* pCtx);
```

DAABlowfishFinal

Completes computation of the DAC value.

```
IppStatus ippsDAABlowfishFinal(Ipp8u *pDAC, int dacLen,
   IppsDAABlowfishState* pCtx);
```

DAABlowfishMessageDigest

Computes the DAC value of the message.

```
IppStatus ippsDAABlowfishMessageDigest(const Ipp8u *pSrcMsg, int msgLen,
   const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int macLen);
```

DAATwofishGetSize

Gets the size of the `IppsDAATwofishState` context.

```
IppStatus ippsDAATwofishGetSize(int *pSize);
```

DAATwofishInit

Initializes user supplied memory as `IppsDAATwofishState` context for future use.

```
IppStatus ippsDAATwofishInit(const Ipp8u* pKey, int keylen,
   IppsDAATwofishState* pCtx);
```

DAATwofishUpdate

Digests the current input message stream of the specified length.

```
IppStatus ippsDAATwofishUpdate(const Ipp8u *pSrcMesg, int mesglen,
   IppsDAATwofishState* pCtx);
```

DAATwofishFinal
Completes computation of the DAC value.

```
IppStatus ippsDAATwofishFinal(Ipp8u *pDAC, int dacLen,
   IppsDAATwofishState* pCtx);
```

DAATwofishMessageDigest
Computes the DAC value of the message.

```
IppStatus ippsDAATwofishMessageDigest(const Ipp8u *pSrcMsg, int msgLen,
   const Ipp8u *pKey, int keyLen, Ipp8u *pMAC, int maclen);
```

Public Key Cryptography Functions

Big Number Arithmetic

Add_BNU
Adds two unsigned integer big numbers of the same length.

```
IppStatus ippsAdd_BNU(const Ipp32u *a, const Ipp32u *b, Ipp32u *r, int n,
   Ipp32u *carry);
```

Sub_BNU
Subtracts one integer big number from another integer big number of the same length.

```
IppStatus ippsSub_BNU(const Ipp32u *a, const Ipp32u *b, Ipp32u *r, int n,
   Ipp32u *carry);
```

MulOne_BNU
Multiplies unsigned integer big number by 32-bit unsigned integer.

```
IppStatus ippsMulOne_BNU(const Ipp32u *a, Ipp32u *r, int n, Ipp32u w,
   Ipp32u *carry);
```

MACOne_BNU_I
Computes multiplication of unsigned integer big number by 32-bit integer and accumulates the result with another integer big number.

```
IppStatus ippsMACOne_BNU_I(const Ipp32u *a, Ipp32u *r, int n, Ipp32u w,
   Ipp32u *carry);
```

Mul_BNU4
Multiplies two unsigned integers of 4*32 bits.

```
IppStatus ippsMul_BNU4(const Ipp32u *a, const Ipp32u *b, Ipp32u *r);
```

Mul_BNU8
Multiplies two unsigned integers of 8*32 bits.

```
IppStatus ippsMul_BNU8(const Ipp32u *a, const Ipp32u *b, Ipp32u *r);
```

Div_64u32u

Divides unsigned 64-bit integer by unsigned 32-bit integer.

```
IppStatus ippsDiv_64u32u(Ipp64u a, Ipp32u b, Ipp32u *q, Ipp32u *r);
```

Sqr_32u64u

Computes the square of 32-bit words in the input array.

```
IppStatus ippsSqr_32u64u(const Ipp32u *src, int n, Ipp64u *dst);
```

Sqr_BNU4

Computes the square of an unsigned integer big number of 4*32 bits.

```
IppStatus ippsSqr_BNU4(const Ipp32u *a, Ipp32u *r);
```

Sqr_BNU8

Computes the square of an unsigned integer big number of 8*32 bits.

```
IppStatus ippsSqr_BNU8(const Ipp32u *a, Ipp32u *r);
```

SetOctString_BNU

Converts octet string into unsigned integer big number.

```
IppStatus ippsSetOctString_BNU(const Ipp8u* pOctStr, int strLen, Ipp32u*
   pBNU, int* pBNUsize);
```

GetOctString_BNU

Converts unsigned integer big number into octet string.

```
IppStatus ippsGetOctString_BNU(const Ipp32u* pBNU, int bnuSize, Ipp8u*
   pOctStr, int strLen);
```

BigNumGetSize

Gets the size of the `IppsBigNumState` context in bytes.

```
IppStatus ippsBigNumGetSize(int length, int *size);
```

BigNumInit

Initializes context and partitions allocated buffer.

```
IppStatus ippsBigNumInit(int length, IppsBigNumState *b);
```

Set_BN

Defines the sign and value of the context.

```
IppStatus ippsSet_BN(IppsBigNumSGN sgn, int length, const Ipp32u *data,
   IppsBigNumState *x);
```

SetOctString_BN

Converts octet string into a positive Big Number.

```
IppStatus ippsSetOctString_BN(const Ipp8u* pOctStr, int strLen,
   IppsBigNumState* pBN);
```

GetSize_BN

Returns the maximum length of the integer big number the structure can store.

```
IppStatus ippsGetSize_BN(const IppsBigNumState *b, int *size);
```

Get_BN

Extracts the sign and value of the integer big number from the input structure.

```
IppStatus ippsGet_BN(IppsBigNumSGN *sgn, int *length, Ipp32u *data, const
   IppsBigNumState *x);
```

GetOctString_BN

Converts a positive Big Number into octet String.

```
IppStatus ippsGetOctString_BN(const Ipp8u* pOctStr, int strLen, const
   IppsBigNumState* pBN);
```

Cmp_BN

Compares two Big Numbers.

```
IppStatus ippsCmp_BN(const IppsBigNumState *pA, const IppsBigNumState
   *pB, Ipp32u *pRsult);
```

CmpZero_BN

Checks the value of the input data field.

```
IppStatus ippsCmpZero_BN(const IppsBigNumState *b, Ipp32u *result);
```

Add_BN

Adds two integer big numbers.

```
IppStatus ippsAdd_BN(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *r);
```

Sub_BN

Subtracts one integer big number from another.

```
IppStatus ippsSub_BN(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *r);
```

Mul_BN

Multiplies two integer big numbers.

```
IppStatus ippsMul_BN(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *r);
```

MAC_BN_I

Multiplies two integer big numbers and accumulates the result with the third integer big number.

```
IppStatus ippsMAC_BN_I(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *r);
```

Div_BN
Divides one integer big number by another.

```
IppStatus ippsDiv_BN(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *q, IppsBigNumState *r);
```

Mod_BN
Computes modular reduction for input integer big number with respect to specified modulus.

```
IppStatus ippsMod_BN(IppsBigNumState *a, IppsBigNumState *m,
   IppsBigNumState *r);
```

Gcd_BN
Computes greatest common divisor.

```
IppStatus ippsGcd_BN(IppsBigNumState *a, IppsBigNumState *b,
   IppsBigNumState *g);
```

ModInv_BN
Computes multiplicative inverse of a positive integer big number with respect to specified modulus.

```
IppStatus ippsModInv_BN(IppsBigNumState *e, IppsBigNumState *m,
   IppsBigNumState *d);
```

Montgomery Reduction Scheme Functions

MontGetSize
Gets the size of the `IppsMontState` context.

```
IppStatus ippsMontGetSize(IppsExpMethod method, int length, int *size);
```

MontInit
Initializes the context and partitions the specified buffer space.

```
IppStatus ippsMontInit(IppsExpMethod method, int length, IppsMontState
   *m);
```

MontSet
Sets the input integer big number to a value and computes the Montgomery reduction index.

```
IppStatus ippsMontSet(const Ipp32u *n, int length, IppsMontState *m);
```

MontGet
Extracts the big number modulus.

```
IppStatus ippsMontGet(Ipp32u *n, int *length, const IppsMontState *m);
```

MontForm
Converts input positive integer big number into Montgomery form.

```
IppStatus ippsMontForm(IppsBigNumState *a, IppsMontState *m,
   IppsBigNumState *r);
```

MontMul

Computes Montgomery modular multiplication for positive integer big numbers of Montgomery form.

```
IppStatus ippsMontMul(IppsBigNumState *a, IppsBigNumState *b,
   IppsMontState *m, IppsBigNumState *r);
```

MontExp

Computes Montgomery exponentiation.

```
IppStatus ippsMontExp(IppsBigNumState *a, IppsBigNumState *e,
   IppsMontState *m, IppsBigNumState *r);
```

Pseudorandom Number Generation Functions

PRNGGetSize

Gets the size of the `IppsPRNGState` context in bytes.

```
IppStatus ippsPRNGGetSize(int *pSize);
```

PRNGInit

Initializes user supplied memory as `IppsPRNGState` context for future use.

```
IppStatus ippsPRNGInit(int seedBits, IppsPRNGState* pCtx);
```

PRNGSetSeed

Sets up the seed value for the pseudorandom number generator.

```
IppStatus ippsPRNGSetSeed(const IppsBigNumState* pSeed, IppsPRNGState*
   pCtx);
```

PRNGSetAugment

Sets the initial state with the given input entropy for the pseudorandom number generation.

```
IppsStatus ippsPRNGSetAugment(const IppsBigNumState* pAugment,
   IppsPRNGState* pCtx);
```

PRNGSetModulus

Sets the initial state with the given input modulus for the pseudorandom number generation.

```
IppStatus ippsPRNGSetModulus(const IppsBigNumState* pModulus,
   IppsPRNGState* pCtx);
```

PRNGSetH0

Sets the initial state with the given input IV for the SHA-1 algorithm.

```
IppStatus ippsPRNGSetH0(const IppsBigNumState* pH0, IppsPRNGState*
   pCtx);
```

PRNGen

Generates a pseudorandom unsigned Big Number of the specified bitlength.

```
IppStatus ippsPRNGen(Ipp32u* pRandBNU, int nBits, void* pCtx);
```

PRNGen_BN

Generates a pseudorandom positive Big Number of the specified bitlength.

```
IppStatus ippsPRNGen_BN(IppsBigNumState* pRandBN, int nBits, void*
   pCtx);
```

Prime Number Generation Functions

PrimeGetSize

Gets the size of the `IppsPrimeState` context in bytes.

```
IppStatus ippsPrimeGetSize(int nMaxBits, int* pSize);
```

PrimeInit

Initializes user supplied memory as `IppsPrimeState` context for future use.

```
IppStatus ippsPrimeInit(int nMaxBits, IppsPrimeState* pCtx);
```

PrimeGen

Generates a random probable prime number of the specified bitlength.

```
IppStatus ippsPrimeGen(int nBits, int nTrials, IppsPrimeState* pCtx,
   IppBitSupplier rndFunc, void* pRndParam);
```

PrimeTest

Tests the given integer for being a probable prime.

```
IppStatus ippsPrimeTest(int nTrials, Ipp32u *pResult, IppsPrimeState*
   pCtx, IppBitSupplier rndFunc, void* pRndParam);
```

PrimeSet

Sets the Big Number for primality testing.

```
IppStatus ippsPrimeSet(const Ipp32u* pBNU, int nBits, IppsPrimeState*
   pCtx);
```

PrimeSet_BN

Sets the Big Number for primality testing.

```
IppStatus ippsPrimeSet_BN(const IppsBigNumState* pBN, IppsPrimeState*
   pCtx);
```

PrimeGet

Extracts the probable prime unsigned integer big number.

```
IppStatus ippsPrimeGet(Ipp32u* pBNU, int *pSize, const IppsPrimeState
   *pCtx);
```

PrimeGet_BN

Extracts the probable prime positive Big Number.

```
IppStatus IppsPrimeGet_BN(IppsBigNumState* pBN, const IppsPrimeState
   *pCtx);
```

RSA Algorithm Functions

Functions for Building RSA System

RSAGetSize

Gets the size of the `IppsRSAState` context.

```
IppStatus ippsRSAGetSize(int nBitsN, int nBitsP, IppsRSAKeyType flag,
   int* pSize);
```

RSAInit

Initializes user supplied memory as the `IppsRSAState` context for future use.

```
IppStatus ippsRSAInit(int nBitsN, int nBitsP, IppsRSAKeyType flag,
   IppsRSAState* pCtx);
```

RSASetKey

Sets the tag-designated key component into the established RSA context.

```
IppStatus ippsRSASetKey(const IppsBigNumState* pBN, ippsRSAKeyTag tag,
   IppsRSAState* pCtx);
```

RSAGetKey

Extracts the tag-designated key component from the RSA context.

```
IppStatus ippsRSAGetKey(IppsBigNumState* pBN, IppsRSAKeyTag tag,
   IppsRSAState* pCtx);
```

RSAGenerate

Generates key components for the desired RSA cryptographic system.

```
IppStatus ippsRSAGenerate(IppsBigNumState* pE, int nBitsN, int nBitsP,
   int nTrials, IppsRSAState* pCtx, IppBitSupplier rndFunc, void*
   pRndParam);
```

RSAValidate

Validates key components of the RSA cryptographic system.

```
IppStatus ippsRSAValidate(const IppsBigNumState* pE, int nTrials,
   Ipp32u* pResult, IppsRSAState* pCtx, IppBitSuppler rndFunc, void*
   pRndParam);
```

RSA Primitives

RSAEncrypt

Performs the RSA encryption operation.

```
IppStatus ippsRSAEncrypt(const IppsBigNumState* pX, IppsBigNumState* pY,
   IppsRSAState* pCtx);
```

RSADecrypt

Performs the RSA decryption operation.

```
IppStatus ippsRSADecrypt(IppsBigNumState* pX, IppsBigNumState* pY,
   IppsRSAState* pCtx);
```

RSA-OAEP Scheme Functions

RSAOAEPEncrypt

Carries out the RSA-OAEP encryption scheme.

```
IppStatus ippsRSAOAEPEncrypt(const Ipp8u* pSrc, int srcLen, const Ipp8u*
   pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst, IppsRSAState*
   pCtx, IppHash hushFunc, int hashLen, IppMGF mgfFunc);
```

RSAOAEPEncrypt_MD5

Carries out the RSA-OAEP encryption scheme using MD5 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_MD5(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPEncrypt_SHA1

Carries out the RSA-OAEP encryption scheme using SHA-1 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_SHA1(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPEncrypt_SHA224

Carries out the RSA-OAEP encryption scheme using SHA-224 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_SHA224(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPEncrypt_SHA256

Carries out the RSA-OAEP encryption scheme using SHA-256 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_SHA256(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPEncrypt_SHA384

Carries out the RSA-OAEP encryption scheme using SHA-384 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_SHA384(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPEncrypt_SHA512

Carries out the RSA-OAEP encryption scheme using SHA-512 hash algorithm.

```
IppStatus ippsRSAOAEPEncrypt_SHA512(const Ipp8u* pSrc, int srcLen, const
   Ipp8u* pLabel, int labLen, const Ipp8u* pSeed, Ipp8u* pDst,
   IppsRSAState* pCtx);
```

RSAOAEPDecrypt

Carries out the RSA-OAEP decryption scheme.

```
IppStatus ippsRSAOAEPDecrypt(const Ipp8u* pSrc, const Ipp8u* pLabel, int
   labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx, IppHash
   hushFunc, int hashLen, IppMGF mgfFunc);
```

RSAOAEPDecrypt_MD5

Carries out the RSA-OAEP decryption scheme using MD5 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_MD5(const Ipp8u* pSrc, const Ipp8u* pLabel,
   int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSAOAEPDecrypt_SHA1

Carries out the RSA-OAEP decryption scheme using SHA-1 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_SHA1(const Ipp8u* pSrc, const Ipp8u*
   pLabel, int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSAOAEPDecrypt_SHA224

Carries out the RSA-OAEP decryption scheme using SHA-224 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_SHA224(const Ipp8u* pSrc, const Ipp8u*
   pLabel, int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSAOAEPDecrypt_SHA256

Carries out the RSA-OAEP decryption scheme using SHA-256 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_SHA256(const Ipp8u* pSrc, const Ipp8u*
   pLabel, int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSAOAEPDecrypt_SHA384

Carries out the RSA-OAEP decryption scheme using SHA-384 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_SHA384(const Ipp8u* pSrc, const Ipp8u*
   pLabel, int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSAOAEPDecrypt_SHA512

Carries out the RSA-OAEP decryption scheme using SHA-512 hash algorithm.

```
IppStatus ippsRSAOAEPDecrypt_SHA512(const Ipp8u* pSrc, const Ipp8u*
   pLabel, int labLen, Ipp8u* pDst, int* pDstLen, IppsRSAState* pCtx);
```

RSA-SSA Scheme Functions

RSASSASign

Carries out the RSA-SSA signature generation scheme.

```
IppStatus ippsRSASSASign(const Ipp8u* pHMsg, int hashLen, const Ipp8u*
   pSalt, int saltLen, Ipp8u* pSign, IppsRSAState* pCtx, IppHash
   hushFunc, IppMGF mgfFunc);
```

RSASSASign_MD5

Carries out the RSA-SSA signature generation scheme using MD5 hash algorithm.

```
IppStatus ippsRSASSASign_MD5(const Ipp8u* pHMsg, const Ipp8u* pSalt, int
   saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSASign_SHA1

Carries out the RSA-SSA signature generation scheme using SHA-1 hash algorithm.

```
IppStatus ippsRSASSASign_SHA1(const Ipp8u* pHMsg, const Ipp8u* pSalt, int
   saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSASign_SHA224

Carries out the RSA-SSA signature generation scheme using SHA-224 hash algorithm.

```
IppStatus ippsRSASSASign_SHA224(const Ipp8u* pHMsg, const Ipp8u* pSalt,
   int saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSASign_SHA256

Carries out the RSA-SSA signature generation scheme using SHA-256 hash algorithm.

```
IppStatus ippsRSASSASign_SHA256(const Ipp8u* pHMsg, const Ipp8u* pSalt,
   int saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSASign_SHA384

Carries out the RSA-SSA signature generation scheme using SHA-384 hash algorithm.

```
IppStatus ippsRSASSASign_SHA384(const Ipp8u* pHMsg, const Ipp8u* pSalt,
   int saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSASign_SHA512

Carries out the RSA-SSA signature generation scheme using SHA-512 hash algorithm.

```
IppStatus ippsRSASSASign_SHA512(const Ipp8u* pHMsg, const Ipp8u* pSalt,
   int saltLen, Ipp8u* pSign, IppsRSAState* pCtx);
```

RSASSAVerify

Carries out the RSA-SSA signature verification scheme.

```
IppStatus ippsRSASSAVerify(const Ipp8u* pHMsg, int hashLen, const Ipp8u*
   pSign, IppBool* pIsValid, IppsRSAState* pCtx, IppHash hushFunc,
   IppMGF mgfFunc);
```

RSASSAVerify_MD5

Carries out the RSA-SSA signature verification scheme using MD5 hash algorithm.

```
IppStatus ippsRSASSAVerify_MD5(const Ipp8u* pHMsg, const Ipp8u* pSign,
   IppBool* pIsValid, IppsRSAState* pCtx);
```

RSASSAVerify_SHA1

Carries out the RSA-SSA signature verification scheme using SHA-1 hash algorithm.

```
IppStatus ippsRSASSAVerify_SHA1(const Ipp8u* pHMsg, const Ipp8u* pSign,
   IppBool* pIsValid, IppsRSAState* pCtx);
```

RSASSAVerify_SHA224

Carries out the RSA-SSA signature verification scheme using SHA-224 hash algorithm.

```
IppStatus ippsRSASSAVerify_SHA224(const Ipp8u* pHMsg, const Ipp8u*
   pSign, IppBool* pIsValid, IppsRSAState* pCtx);
```

RSASSAVerify_SHA256

Carries out the RSA-SSA signature verification scheme using SHA-256 hash algorithm.

```
IppStatus ippsRSASSAVerify_SHA256(const Ipp8u* pHMsg, const Ipp8u*
   pSign, IppBool* pIsValid, IppsRSAState* pCtx);
```

RSASSAVerify_SHA384

Carries out the RSA-SSA signature verification scheme using SHA-384 hash algorithm.

```
IppStatus ippsRSASSAVerify_SHA384(const Ipp8u* pHMsg, const Ipp8u*
   pSign, IppBool* pIsValid, IppsRSAState* pCtx);
```

RSASSAVerify_SHA512

Carries out the RSA-SSA signature verification scheme using SHA-512 hash algorithm.

```
IppStatus ippsRSASSAVerify_SHA512(const Ipp8u* pHMsg, const Ipp8u*
   pSign, IppBool* pIsValid, IppsRSAState* pCtx);
```

Discrete-Logarithm-Based Cryptography Functions

DLPGetSize

Gets the size of the `IppsDLPState` context.

```
IppStatus ippsDLPGetSize(int peBits, int reBits, int *pSize);
```

DLPInit

Initializes user supplied memory as the `IppsDLPState` context for future use.

```
IppsStatus IppsDLPInit(int peBits, int reBits, IppsDLPState* pCtx);
```

DLPSet

Sets up domain parameters of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPSet(const IppsBigNumState* pP, const IppBigNumState*
   pQ, const IppsBigNumState* pG, IppsDLPState* pCtx);
```

DLPGet

Retrieves domain parameters of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPGet(IppsBigNumState* pP, IppsBigNumState* pQ,
   IppsBigNumState* pG, ippsDLPState* pCtx);
```

DLPSetDP

Sets up a particular domain parameter of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPSetDP(const IppsBigNumState* pDP, IppsDLPKeyTag tag,
   IppsDLPState* pCtx);
```

DLPGetDP

Retrieves a particular domain parameter of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPGetDP(const IppsBigNumState* pDP, IppsDLPKeyTag tag,
   IppsDLPState* pCtx);
```

DLPGenKeyPair

Generates a private key and computes public keys of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPGenKeyPair(ippsBigNumState* pPrivate, ippsBigNumState*
   pPublic, IppsDLPState* pCtx, IppBitSupplier rndFunc, void*
   pRndParam);
```

DLPPublicKey

Computes a public key from the given private key of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPPublicKey(const IppsBigNumState* pPrivate,
   IppsBigNumState* pPublic, IppsDLPState* pCtx);
```

DLPValidateKeyPair

Validates private and public keys of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPValidateKeyPair(const IppsBigNumState* pPrivate, const
   IppsBigNumState* pPublic, IppDLPResult* pResult, IppsDLPState* pCtx);
```

DLPSetKeyPair

Sets private and/or public keys of the DL-based cryptosystem over GF(p).

```
IppStatus ippsDLPSetKeyPair(const IppsBigNumState* pPrivate, const
   IppsBigNumState* pPublic, IppsDLPState* pCtx);
```

DLPGenerateDSA

Generates domain parameters of the DL-based cryptosystem over GF(p) to use DSA.

```
ippStatus ippsDLPGenerateDSA(const IppsBigNumState* pSeedIn, int
   nTrials, IppsDLPState* pCtx, IppsBigNumState* pSeedOut, int*
   pCounter, IppBitSupplier rndFunc, void* pRndParam);
```

DLPValidateDSA

Validates domain parameters of the DL-based cryptosystem over GF(p) to use DSA.

```
IppStatus ippsDLPValidateDSA(int nTrials, IppDLResult* pResult,
   IppsDLPState* pCtx, IppBitSupplier rndFunc, void* pRndParam);
```

DLPSignDSA

Performs the DSA digital signature signing operation.

```
IppStatus ippsDLPSignDSA(const IppsBigNumState* pMsg, const
   IppsBigNumState* pPrivate, IppsBigNumState* pSignR, IppsBigNumState*
   pSignS, IppsDLPState* pCtx);
```

DLPVerifyDSA

Verifies the input DSA digital signature.

```
IppStatus ippsDLPVerifyDSA(const IppsBigNumState* pMsg, const
    IppsBigNumState* pSignR, const IppsBigNumState* pSignS, IppDLResult*
    pResult, IppsDLPState* pCtx);
```

DLPGenerateDH

Generates domain parameters of the DL-based cryptosystem over GF(p) to use the DH Agreement scheme.

```
IppStatus IppsDLPGenerateDH(const IppsBigNumState* pSeedIn, int nTrials,
    IppsDLPState* pCtx, IppsBigNumState* pSeedOut, int* pCounter,
    IppBitSupplier rndFunc, void* pRndParam);
```

DLPValidateDH

Validates domain parameters of the DL-based cryptosystem over GF(p) to use the DH Agreement scheme.

```
IppStatus ippsDLPValidateDH(int nTrials, IppDLResult* pResult,
    IppsDLPState* pCtx, IppBitSupplier rndFunc, void* pRndParam);
```

DLPSharedSecretDH

Computes a shared field element by using the Diffie-Hellman scheme.

```
IssStatus ippsDLPSharedSecretDH(const IppsBigNumState* pPrivateA, const
    IppsBigNumState* pPublicB, IppsBigNumState* pShare, IppsDLPState*
    pCtx);
```

Elliptic Curve Cryptography Functions

Functions Based on GF(*p*)

ECCPGetSize

Gets the size of the `IppsECCPState` context.

```
IppStatus ippsECCPGetSize(int feBitSize, int *pSize);
```

ECCPInit

Initializes context for the elliptic curve cryptosystem over GF(p).

```
IppStatus ippsECCPInit(int feBitSize, IppsECCPState* pECC);
```

ECCPSet

Sets up elliptic curve domain parameters over GF(p).

```
IppStatus ippsECCPSet(const IppsBigNumState* pPrime, const
    IppsBigNumState* pA, const IppsBigNumState* pB, const
    IppsBigNumState* pGX, const IppsBigNumState* pGY, const
    IppsBigNumState* pOrder, int cofactor, IppsECCPState* pECC);
```

ECCPSetStd

Sets up a recommended set of elliptic curve domain parameters over GF(*p*).

```
IppStatus ippsECCPSetStd(IppECCType flag, IppsECCPState* pECC);
```

ECCPGet

Retrieves elliptic curve domain parameters over GF(p).

```
IppStatus ippsECCPGet(IppsBigNumState* pPrime, IppsBigNumState* pA,
   IppsBigNumState* pB, IppsBigNumState* pGX,IppsBigNumState* pGY,
   IppsBigNumState* pOrder, int* cofactor,  IppsECCPState* pECC);
```

ECCPGetOrderBitSize

Retrieves order size of the elliptic curve base point over GF(p) in bits.

```
IppStatus ippsECCPGetOrderBitSize(int* pBitSize, IppsECCPState* pECC);
```

ECCPValidate

Checks validity of the elliptic curve domain parameters over GF(p).

```
IppStatus ippsECCPValidate(int nTrials, IppECResult* pResult,
   IppsECCPState* pECC, IppBitSupplier rndFunc, void* pRndParam);
```

ECCPPointGetSize

Gets the size of the `IppsECCPPoint` context in bytes for a point on the elliptic curve point defined over GF(*p*).

```
IppStatus ippsECCPPointGetSize(int feBitSize, int* pSize);
```

ECCPPointInit

Initializes the context for a point on the elliptic curve defined over GF(p).

```
IppStatus ippsECCPPointInit(int feBitSize, IppsECCPPoint* pPoint);
```

ECCPSetPoint

Sets coordinates of a point on the elliptic curve defined over GF(p).

```
IppStatus ippsECCPSetPoint(const IppsBigNumState* pX, const
   IppsBigNumState* pY, IppsECCPPoint* pPoint, IppsECCPState* pECC);
```

ECCPSetPointAtInfinity

Sets the point at infinity.

```
IppStatus ippsECCPSetPointAtInfinity(IppsECCPPoint* pPoint,
   IppsECCPState* pECC);
```

ECCPGetPoint

Retrieves coordinates of the point on the elliptic curve defined over GF(p).

```
IppStatus ippsECCPGetPoint(IppsBigNumState* pX, IppsBigNumState* pY,
   const IppsECCPPoint* pPoint, IppsECCPState* pECC);
```

ECCPCheckPoint

Checks *correctness of the point on the elliptic curve defined over GF(*p*).*

```
IppStatus ippsECCPCheckPoint(const IppsECCPPoint* pP, IppECResult*
   pResult, IppsECCPState* pECC);
```

ECCPComparePoint

Compares two points on the elliptic curve defined over GF(p).

```
IppStatus ippsECCPComparePoint(const IppsECCPPoint* pP, const
   IppsECCPPoint* pQ, IppECResult* pResult, IppsECCPState* pECC);
```

ECCPNegativePoint

Finds an elliptic curve *point which is an additive inverse for the given point over GF(p).*

```
IppStatus ippsECCPNegativePoint(const IppsECCPPoint* pP, IppsECCPPoint*
   pR, IppsECCPState* pECC);
```

ECCPAddPoint

Computes the addition of two elliptic curve points over GF(p).

```
IppStatus ippsECCPAddPoint(const IppsECCPPoint* pP, const IppsECCPPoint*
   pQ, IppsECCPPoint* pR, IppsECCPState* pECC);
```

ECCPMulPointScalar

Performs scalar multiplication of a point on the elliptic curve defined over GF(p).

```
IppStatus ippsECCPMulPointScalar(const IppsECCPPoint* pP, const
   IppsBigNumState* pK, IppsECCPPoint* pR, IppsECCPState* pECC);
```

ECCPGenKeyPair

Generates a private key and computes public keys of the elliptic cryptosystem over GF(p).

```
IppStatus ippsECCPGenKeyPair(IppsBigNumState* pPrivate,
   IppsECCPPointState* pPublic, IppsECCPState* pECC, IppBitSupplier
   rndFunc, void* pRndParam);
```

ECCPPublicKey

Computes a public key from the given private key of the elliptic cryptosystem over GF(p).

```
IppStatus ippsECCPPublicKey(const IppsBigNumState* pPrivate,
   IppsECCPPoint* pPublic, IppsECCPState* pECC);
```

ECCPValidateKeyPair

Validates private and public keys of the elliptic cryptosystem over GF(p).

```
IppStatus ippsECCPValidateKeyPair(const IppsBigNumState* pPrivate, const
   IppsECCPPoint* pPublic, IppECResult* pResult, IppsECCPState* pECC);
```

ECCPSetKeyPair

Sets private and/or public keys of the elliptic cryptosystem over GF(p).

```
IppStatus ippsECCPSetKeyPair(const IppsBigNumState* pPrivate, const
   IppsECCPPoint* pPublic, IppBool regular, IppsECCPState* pECC);
```

ECCPSharedSecretDH

Computes a shared secret field element by using the Diffie-Hellman scheme.

```
IppStatus ippsECCPSharedSecretDH(const IppsBigNumState* pPrivate, const
   IppsECCPPointState* pPublic, IppsBigNumState* pShare,
   IppsECCPState* pECC);
```

ECCPSharedSecretDHC

Computes a shared secret field element by using the Diffie-Hellman scheme and the elliptic curve cofactor.

```
IppStatus ippsECCPSharedSecretDHC(const IppsBigNumState* pPrivate, const
   IppsECCPPointState* pPublic, IppsBigNumState* pShare,  IppsECCPState*
   pECC);
```

ECCPSignDSA

Computes a digital signature over a message digest.

```
IppStatus ippsECCPSignDSA(const IppsBigNumState* pMsgDigest, const
   IppsBigNumState* pPrivate, IppsBigNumState* pSignX, IppsBigNumState*
   pSignY, IppsECCPState* pECC);
```

ECCPVerifyDSA

Verifies authenticity of the digital signature over a message digest (ECDSA).

```
IppStatus ippsECCPVerifyDSA(const IppsBigNumState*  pMsgDigest, const
   IppsBigNumState*  pSignX, const IppsBigNumState*  pSignY,
   IppECResult* pResult, IppsECCPState* pECC);
```

ECCPSignNR

Computes the digital signature over a message digest (the Nyberg-Rueppel scheme).

```
IppStatus ippsECCPSignNR(const IppsBigNumState* pMsgDigest,
   IppsBigNumState* pSignX, IppsBigNumState* pSignY, IppsECCPState*
   pECC);
```

ECCPVerifyNR

Verifies authenticity of the digital signature over a message digest (the Nyberg-Rueppel scheme).

```
IppStatus ippsECCPVerifyNR(const IppsBigNumState*  pMsgDigest, const
   IppsBigNumState*  pSignX, const IppsBigNumState*  pSignY,
   IppECResult* pResult,  IppsECCPState* pECC);
```

Functions Based on GF(2^m)

ECCBGetSize

Gets the size of the `IppsECCBState` context.

```
IppStatus ippsECCBGetSize(int feBitSize, int *pSize);
```

ECCBInit

Initializes context for the elliptic curve cryptosystem over GF(2^m).

```
IppStatus ippsECCBInit(int feBitSize, IppsECCBState* pECC)
```

ECCBSet

Sets up elliptic curve domain parameters over GF(2^m).

```
IppStatus ippsECCBSet(const IppsBigNumState* pPrime, const
   IppsBigNumState* pA, const IppsBigNumState* pB, const
   IppsBigNumState* pGX, const IppsBigNumState* pGY, const
   IppsBigNumState* pOrder, int cofactor, IppsECCBState* pECC);
```

ECCBSetStd

Sets up a recommended set of elliptic curve domain parameters over GF(2^m).

```
IppStatus ippsECCBSetStd(IppECCType flag, IppsECCBState* pECC);
```

ECCBGet

Retrieves elliptic curve domain parameters over GF(2^m).

```
IppStatus ippsECCBGet(IppsBigNumState* pPrime, IppsBigNumState* pA,
   IppsBigNumState* pB, IppsBigNumState* pGX,IppsBigNumState* pGY,
   IppsBigNumState* pOrder, int* cofactor,  IppsECCBState* pECC);
```

ECCBGetOrderBitSize

Retrieves order size of the elliptic curve base point over GF(2^m) in bits.

```
IppStatus ippsECCBGetOrderBitSize(int* pBitSize, IppsECCBState* pECC);
```

ECCBValidate

Checks validity of the elliptic curve domain parameters over GF(2^m).

```
IppStatus ippsECCBValidate(int nTrials, IppECResult* pResult,
   IppsECCBState* pECC, IppBitSupplier rndFunc, void* pRndParam);
```

ECCBPointGetSize

Gets the size of the `IppsECCBPoint` context in bytes for a point on the elliptic curve point defined over GF(2^m).

```
IppStatus ippsECCBPointGetSize(int feBitSize, int* pSize);
```

ECCBPointInit

Initializes the context for a point on the elliptic curve defined over GF(2^m).

```
IppStatus ippsECCBPointInit(int feBitSize, IppsECCBPoint* pPoint);
```

ECCBSetPoint

Sets coordinates of a point on the elliptic curve defined over GF(2^m).

```
IppStatus ippsECCBSetPoint(const IppsBigNumState* pX, const
   IppsBigNumState* pY, IppsECCBPoint* pPoint, IppsECCBState* pECC);
```

ECCBSetPointAtInfinity

Sets the point at infinity.

```
IppStatus ippsECCBSetPointAtInfinity(IppsECCBPoint* pPoint,
   IppsECCBState* pECC);
```

ECCBGetPoint

Retrieves coordinates of the point on the elliptic curve defined over GF(2^m).

```
IppStatus ippsECCBGetPoint(IppsBigNumState* pX, IppsBigNumState* pY,
   const IppsECCBPoint* pPoint, IppsECCBState* pECC);
```

ECCBCheckPoint

Checks *correctness of the point on the elliptic curve defined over GF(2^m).*

```
IppStatus ippsECCBCheckPoint(const IppsECCBPoint* pP, IppECResult*
   pResult, IppsECCBState* pECC);
```

ECCBComparePoint

Compares two points on the elliptic curve defined over GF(2^m).

```
IppStatus ippsECCBComparePoint(const IppsECCBPoint* pP, const
   IppsECCBPoint* pQ, IppECResult* pResult, IppsECCBState* pECC);
```

ECCBNegativePoint

Finds the elliptic curve *point which is an additive inverse for the given point over GF(2^m).*

```
IppStatus ippsECCBNegativePoint(const IppsECCBPoint* pP, IppsECCBPoint*
   pR, IppsECCBState* pECC);
```

ECCBAddPoint

Computes the addition of two elliptic curve points over GF(2^m).

```
IppStatus ippsECCBAddPoint(const IppsECCBPoint* pP, const IppsECCBPoint*
   pQ, IppsECCBPoint* pR, IppsECCBState* pECC);
```

ECCBMulPointScalar

Performs scalar multiplication of a point on the elliptic curve defined over GF(2^m).

```
IppStatus ippsECCBMulPointScalar(const IppsECCBPoint* pP, const
   IppsBigNumState* pK, IppsECCBPoint* pR, IppsECCBState* pECC);
```

ECCBGenKeyPair

Generates a private key and computes public keys of the elliptic cryptosystem over GF(2^m).

```
IppStatus ippsECCBGenKeyPair(IppsBigNumState* pPrivate,
   IppsECCBPointState* pPublic, IppsECCBState* pECC, IppBitSupplier
   rndFunc, void* pRndParam);
```

ECCBPublicKey

Computes a public key from the given private key of the elliptic cryptosystem over GF(2^m).

```
IppStatus ippsECCBPublicKey(const IppsBigNumState*  pPrivate,
   IppsECCBPoint* pPublic, IppsECCBState* pECC);
```

ECCBValidateKeyPair

Validates private and secret keys of the elliptic cryptosystem over GF(2^m).

```
IppStatus ippsECCBValidateKeyPair(const IppsBigNumState* pPrivate, const
   IppsECCBPoint* pPublic, IppECResult* pResult, IppsECCBState* pECC);
```

ECCBSetKeyPair

Sets private and/or public keys in the elliptic cryptosystem over GF(2^m).

```
IppStatus ippsECCBSetKeyPair(const IppsBigNumState* pPrivate, const
   IppsECCBPoint* pPublic, IppBool regular, IppsECCBState* pECC);
```

ECCBSharedSecretDH

Computes a shared secret field element by using the Diffie-Hellman scheme.

```
IppStatus ippsECCBSharedSecretDH(const IppsBigNumState* pPrivate, const
   IppsECCBPointState* pPublic, IppsBigNumState* pShare, IppsECCBState*
   pECC);
```

ECCBSharedSecretDHC

Computes a shared secret field element by using the Diffie-Hellman scheme and the elliptic curve cofactor.

```
IppStatus ippsECCBSharedSecretDHC(const IppsBigNumState* pPrivate, const
   IppsECCBPointState* pPublic, IppsBigNumState* pShare,  IppsECCBState*
   pECC);
```

ECCBSignDSA

Computes a digital signature over a message digest.

```
IppStatus ippsECCBSignDSA(const IppsBigNumState* pMsgDigest, const
   IppsBigNumState* pPrivate, IppsBigNumState* pSignX, IppsBigNumState*
   pSignY, IppsECCBState* pECC);
```

ECCBVerifyDSA

Verifies authenticity of the digital signature over a message digest (ECDSA).

```
IppStatus ippsECCBVerifyDSA(const IppsBigNumState*  pMsgDigest, const
   IppsBigNumState*  pSignX, const IppsBigNumState*  pSignY,
   IppECResult* pResult, IppsECCBState* pECC);
```

ECCBSignNR

Computes the digital signature over a message digest (the Nyberg-Rueppel scheme).

```
IppStatus ippsECCBSignNR(const IppsBigNumState* pMsgDigest,
   IppsBigNumState* pSignX, IppsBigNumState* pSignY, IppsECCBState*
   pECC);
```

ECCBVerifyNR

Computes authenticity of the digital signature over a message digest (the Nyberg-Rueppel scheme).

```
IppStatus ippsECCBVerifyNR(const IppsBigNumState*  pMsgDigest, const
   IppsBigNumState*  pSignX, const IppsBigNumState*  pSignY,
   IppECResult* pResult,  IppsECCBState* pECC);
```